Science of Computer Programming 18 (1992) 205–222
Elsevier



# Book Reviews*

**Anne Kaldewaij, *Programming: The Derivation of Algorithms* (Prentice-Hall, Hemel Hempstead, United Kingdom, 1990), ISBN 0-13-204108-1 (paperback).**

Kaldewaij's book supports a one-year programming course for first-year students who have just a little exposure to procedural programming already: "(just) enough to have some idea about programs and program execution". From that little exposure, however, the students are resolutely guided towards rigorous derivation.

The approach is in the tradition of weakest-precondition-based texts. Programs are derived from specifications (of Dijkstra and Feijen [1]) based on the triples

$$\{pre\}program\{post\}$$

originally of Hoare. As is now common, however, the triples express total rather than only partial correctness.

Since the *pre* and *post* are predicate formulae, the book introduces the predicate calculus in its own right. And it is clear that much care has been taken to make the subsequent program derivations go smoothly. For example, the self-imposed discipline

> We will always use fresh names for dummies. In particular, program variables will never occur as the name of a bound variable.

allows the simple rule "(syntactic) substitution distributes over all other operators (including quantification)". That avoids entirely the issue of variable capture by quantifiers—at least during derivation of programs—and indeed "variable capture" does not appear in the index. (A drawback, however, is that the issue *does* arise in the use of the predicate calculus more generally, and graduates of the predicate calculus chapter might therefore run into difficulty in later life.)

The program derivations are indeed largely calculations over predicates, rather than over programs themselves, as is still normal with procedural programming. The techniques are at first divided into the well-known general groups—*take a conjunct as invariant*, *replace a constant by a variable*, etc.—and each is nicely illustrated by example. Then later chapters are organised instead by application—for

* Review copies of books which might be of interest to the readers of *Science of Computer Programming* should be sent to Prof. D. Bjoerner (address: see inside front cover). Proceedings of conferences will not normally be reviewed.

Elsevier Science Publishers B.V.

example, *searching, segment optimisations, sorting*—and the general techniques are deployed as appropriate within each application.

Concluding chapters address quite difficult problems, for an introductory course, such as *largest square under a histogram* and *longest common subsequence.* (The *largest* rectangle *under a histogram* is given as an exercise for the reader.)

There is a discussion of program efficiency, and a very brief explanation of the $\mathcal{O}$-notation. That brevity seems appropriate in this text; and phrases like "has at least time complexity $\mathcal{O}(N)$" avoid introducing the otherwise necessary $\Omega$-notation as well. But perhaps too brief is the remark, concerning *Quicksort*, that a linear algorithm exists for median-finding: indeed it does, but its high constant factor makes it inappropriate for use in selecting the pivot. In fact, even the elegant *Dutch National Flag* algorithm is better avoided in *Quicksort* for the same reason: it is linear, but still does "too many" swaps.

Refreshingly for introductory texts in this tradition, even quadratic algorithms are considered worthy of study. And at least one example (*shortest twice-maximal segment*) appears to be quadratic but is shown by detailed analysis still to be linear. (The word "amortized" should have been mentioned.)

A little puzzling is the treatment of assignment to arrays which, rather than define $h.E := F$ (more conventionally, $h[E] := F$) to be the *statement* $h := h(E : F)$, instead gives its meaning as a Hoare-triple directly. That leads to a prohibition of *multiple* assignments containing arrays on the left, even if no two of the variables are the same array. As a result, in the *longest common subsequence* problem the entirely blameless statement

$$a, h.(n+1) := h.(n+1), 1+a$$

is banned, and a temporary variable introduced to express it more primitively.

There are many exercises, and (some) answers are available from the publisher.

The index is quite short (it does not contain *multiple assignment*, for example), and slightly uneven (*selection* appears, but *repetition* does not).

One's impression after reading the book is that its author is a gifted teacher of this material, with a no-nonsense attitude to the role of strict formality. The book's organisation is exceptionally good.

For example, there is a chapter *mixed problems*, roughly half-way through, that provides not only a chance to gather the threads of the earlier material before proceeding to the more difficult problems, but defeats deliberately the meta-reasoning that runs "this exercise is in a section on tail invariants (say), therefore that must be the correct approach to its solution". Excessive formal manipulations are avoided in the later examples by identifying and reusing problem- and solution-schemes from earlier examples, instantiating them in various ways.

Consistent with its aim to be an introduction, the book concentrates on methods of derivation rather than programming language features, and so treats neither procedures nor recursion. Data-refinement is only very briefly and incidentally mentioned. The data structures are limited only to arrays, even using (for example

in non-recursive *Quicksort*) a pair of arrays rather than the more natural array of pairs.

This is an excellent text; it is especially distinguished in the large-scale organisation of chapters, yet it spares nothing in its attention to detail. And it is *short*. But most importantly, it achieves what all these programming texts strive for: it makes the reader feel, afterwards, that he can make computer programs that before he would not have known how to begin.

**Reference**

[1] E.W. Dijkstra and W.H.J. Feijen, *A Method of Programming* (Addison-Wesley, Reading, MA, 1988).

Carroll MORGAN
*Programming Research Group*
*Oxford University*
*Oxford, United Kingdom*

**The Office of Charles and Ray Eames, *A Computer Perspective: Background to the Computer Age—New Edition* (Harvard University Press, Cambridge, MA, 1990), 175 pages, ISBN 0-674-15626-9 (paperback).**

Nothing is as dusty as a dusty computer. Museums have all sorts of problems with their computer departments. It is impossible to properly display the working of the computer—one cannot see its operation because of the small size, the high rate and the purely electronic nature of the events. The important aspect of automata and computers is the dynamics and this aspect is difficult to demonstrate. Some museums, particularly in the United States, offer dynamic programs, mainly computer graphics and computer games, transforming thereby the computer department into a children's section which is only a part of the purpose.

Is the dynamic aspect of the automation, is the history of computing bound to be neglected? There is a general management mentality in this direction: the computer is seen so much as an innovation object that yesterday has no importance. But this is a fundamental double-error. Each technical object (like each natural object) carries the past—the history of its development—with it and many features cannot be understood (and, consequently, not properly applied) without the knowledge of the history. Secondly, a structure as complicated as the computer tends toward intransparency because even the specialist is familiar only with partial aspects. In the early days, things were bigger, slower, and structurally simpler, everything was closer together and it was much easier to have an overview and to follow the development. Studying the early achievements or at least glancing at the history is a key means to understand the present of information processing and the only means to get a prospect of the future. Museums should be seen as a part of school and university—a connex realized only by very few generalists and denied by the majority.